06

Checked y unchecked

Transcripción

[00:00] Bien, ya hemos visto cómo es el ciclo de propagación de una excepción, la jerarquía de las excepciones en Java, la división en los errores y las excepciones como tal y la diferencia que tiene extender de RunTime o de excepción. Ahora tenemos también aquí otro concepto más, un último concepto, que es el más sencillo de todos. Vamos a pasar un poco de positiva. Es el de la clasificación de las excepciones propiamente dichos.

[00:00] Ya sabemos que entre RuntimeException y Exception, la diferencia es que en una yo necesito decirle en la firma el método que con certeza va a lanzar esa excepción y en el caso de RuntimeException es que de repente por ahí él lanza esa excepción.

[00:56] En Java ellos tienen un nombre que es unchecked y checked. Unchecked es el grupo de RuntimeException. ¿Qué significa? Como su nombre lo dice, unchecked, no verificado. Esas excepciones no son verificadas por el compilador. Puede que lance, puede que no nace, el compilador no nos va a alertar si es que por si acaso, él lanzó, él puede lanzar o no puede lanzar, el compilador no va a hacer este trabajo.

[01:27] En el caso de checked, de verificado, el compilador sí nos va a obligar a tratar esa excepción, si es que es lanzada en alguna parte del código. Si no, no va a compilar. Y eso lo comprobamos aquí. ¿Por qué? Porque nuevamente yo voy a borrar este código. Esto es solamente a modo de repaso, solamente para afianzar el el concepto de Exception de checked y unchecked.

[01:53] Vemos claramente aquí que si yo aquí en método 2, yo estoy lanzando explícitamente mi exception, él ya me obliga primero o hacer el try/catch a este nivel, a tratarlo aquí y encerrar la bomba aquí, o avisarle al método que está abajo de la pila: "oye, por si acaso hay una excepción aquí de tal tipo, sí o sí. Tienes que saber cómo tratar con ella".

[02:20] Es la primera diferencia y el compilador ya nos va a avisar en ese caso. Entonces como yo ya estoy avisando en la firma el método, él lanza MiException, entonces automáticamente si yo no lo atrapo aquí, si yo no le doy tratamiento aquí, y también le aviso al de arriba: "Oye, por si acaso lanza MiException".

[02:40] Es como una especie de teléfono entre los entre los métodos de la pila, porque si tú no vas a hacer nada con el Exception, avísale al siguiente que por si acaso le va a llegar un objeto tipo MiException, una bomba tipo MiException. Si él no hace nada, avísale al que sigue, "por si acaso te van a mandar una bomba tipo MiException" hasta que alguien se haga cargo.

[03:03] Y eso es lo que pasa aquí, porque él fue lanzado en método 2. En método 1 nadie le hizo caso, pero sí le avisaron a main que tenía que hacer algo con esa bomba, tenía que hacer algo o la bomba iba a detonar, o mejor dicho, ni siquiera iba a compilar si es que él no hacía nada.

[03:21] Entonces, esa es la diferencia, es una excepción del tipo checked, verificada, verificada por el compilador. El compilador no me deja seguir adelante, ni siquiera generar el bytecode, si es que yo no trato esta excepción, a diferencia de cómo vimos anteriormente, de Runtime. Porque en Runtime yo no necesito avisar nada.

[03:46] Incluso yo tengo aquí: "throws MiException", "throw new MiException", etcétera, y el compilador está feliz por su lado porque dice: "Esa excepción extiende de Runtime entonces es una excepción unchecked, no verificada, entonces no me voy a tomar el trabajo pues de verificar obviamente si él está

atrapando o no". Entonces esta es la diferencia entre estos dos subgrupos de excepciones.

[04:15] Repasando aquí. Throwable en la cabeza tiene dos grandes grupos de errores: errores y excepciones. Los errores son lanzados por los programadores que mantienen la máquina virtual de Java, la JVM, y las excepciones son lanzadas por nosotros que programamos sobre la JVM, la máquina virtual de Java. Dentro de este pequeño grupo de excepciones tenemos dos subgrupos: excepciones chequed y unchequed.

[04:45] Las unchecked son las que extienden de RuntimeException, y por lo tanto las que no son verificadas por el compilador. Contrario a checked, que extienden de Exception y sí son verificadas en el momento pues de hacer la compilación. Entonces en resumen lo único que diferencia es al momento de la compilación, si tú quieres que el compilador te ayude a decidir cómo tratar la excepción y si tú quieres atrapar el error porque tú crees que pueda dar ese error.

[05:21] Ya es una elección dependiendo también del método que estés programando, el flujo de negocios que estés programando, ya son varios factores. Pero básicamente la diferencia es a la hora de compilar. Yo les recomiendo también dar una pequeña hojeada por ahí por las clases que extienden de RuntimeException y de Exception. Conozcan los errores que la JVM ya trae consigo.

[05:49] Conozcan pues el código que la JVM incluye. En el caso de RuntimeException ya vimos que él incluye ArithmeticException y NullPointerException. En el caso de Exception, él tiene clases hijas como IOException, que es cuando trabajamos con archivos.

[06:05] Entonces sí es recomendable tener conocimiento de estos dos grupos de excepciones en Java, cosa de que cuando trabajamos releyendo archivos o con ciertas operaciones, ya hay excepciones preparadas y ya tenemos que

saber cómo es que funciona esto. Aparte, en el examen de certificación son preguntas que vienen muy a menudo.

[06:30] El tema de excepciones en el Centro de Certificación son muy tocadas, son muy bien evaluadas. Entonces cualquier duda pueden postearla en el foro, yo trato de responder lo más rápido posible. Y nos vemos ya en la siguiente clase. Practiquen mucho, nos vemos.